IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT APPLICATION

Inventors: Pramod V. N. Koppol.

Case: Koppol 2

Serial No.: 10/033,512 **Group Art Unit:** 2662

Filing Date: December 27, 2001

Examiner: Levitan, Dmitry

Title: Apparatus and Method for Distributed Software Implementation of OSPF

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

DECLARATION UNDER 37 C.F.R. 1.131

1. I, Pramod V. N. Koppol, received B.E. in Computer Science and Engineering from Osmania University, Hyderabad, India and an M.S. in Computer Science from Southern Illinois University at Carbondale, IL, and a Ph.D from North Carolina State University, Raleigh, NC. I am a Technical Manager in the Networking Research Center at Bell Laboratories, Lucent Technologies Inc, Holmdel, NJ, where I have been doing research on data networks. I have been working in data networking

research since 1996. This work has included network system & protocol design/development, IP/MPLS network traffic engineering tools and network/system security.

I, the Declarant, Pramod V. N. Koppol, state further that:

2. In the above-referenced application, Pramod V. N. Koppol, made an inventive contribution to the invention as recited in claims 1-19.

3. Prior to August 22, 2001 the invention corresponding to the present patent application was conceived and a patent submission in the form of a paper and PowerPoint presentation was made to the patent department of Lucent Technologies in Holmdel, NJ which served as the basis for this patent application. The paper, entitled "Scaling OSPF to Very Large Areas", details the invention described in claims 1-19. As can be seen, for example, at Fig. 7 and its accompanying description – the substantially independent operation of a delegate port card in relation to the invention is described in detail.

4. Prior to August 22, 2001, the submission was received by the patent department of Lucent Technologies and was assigned submission 124380. (See Exhibit B – Submission Information letter to inventor). A submission status printout which gives brief details of the patent submission (including submitter information and brief description) and having a status date of 8/21/2001 is included as Exhibit C. The submission status printout clearly demonstrates the existence of the patent submission and invention prior to August 22, 2001.

5. A confirming memo which acknowledges the opening of Submission IDS # 124380 and which is addressed to the supervisors of the inventors is included as Exhibit D.

Diligence Established

6. Prior to August 22, 2001 up until the filing date of the application on December 27, 2001, I had several discussions with Matt Hodulik, the Lucent patent attorney assigned to prepare the application, to discuss the invention and the drafts of the application.

7. Herein, I certify that all statements made of my own knowledge are true and that all statements made on information and belief are believed to be true. I also understand that willful false statements and the like are punishable by fine, imprisonment or both under 18 U.S.C. 1001 and that willful false statements and the like may jeopardize the validity of the application-at-issue or any patent issuing thereon.

Date: 3/15/06
March 15, 2006

Pramod V. N. Koppol

8/21/01

Scaling OSPF to Very Large Areas

Pramod V. Koppol

Bell Laboratories
Lucent Technologies
101 Crawfords Corner Road
Holmdel, NJ 07733, USA

pramod@lucent.com

Abstract

We consider the problem of realizing the implementation of the OSPF routing protocol [1] on a networking device with a distributed hardware architecture. A distributed hardware architecture typically comprises of a main controller card and a number of port cards each having a reasonably powerful CPU. The main controller card and the port cards do not have any shared memory and have an internal communication mechanism. It is common practice to have implementations of control plane protocols execute on the main controller card and relegate local CPU intensive functions such as processing OA&M functionality to the port cards. Such a hardware architecture could provide substantial improvements in the data plane capability of a networking device without necessarily providing much gain in the control plane software scaleability. In this paper, we propose a mechanism, the OSPF flooding proxy mechanism, for taking advantage of the distributed hardware architecture to achieve a highly scaleable OSPF implementation capable of supporting a large number of nodes in an area. Given the widespread interest in MPLS explicit route based traffic engineering within an autonomous system, and given that most TE mechanisms work best when complete network topology is available, such an OSPF implementation is highly desirable. Also, the next generation terabit router architectures with multiple levels of processor hierarchies and spanning multiple shelves make such protocol implementations very compelling. Note also that while this paper only addresses link state routing issues, the general ideas can be used for other protocols as well.

I. Introduction

OSPF is a link state routing protocol. Adjacent devices exchange information in the form of link state advertisements in such a way that all nodes in a network have a consistant link state database at their disposal. Each node then uses this link state database to make routing decisions. In order to avoid faulty routing, it is imperative that all nodes converge to a common view of the network and each node make routing decisions in a manner agreed to by the rest of the nodes in the network. To achive the first objective, OSPF defines procedures for reliable flooding information originated by any node to the rest of the network. The second objective is achieved in OSPF by mandating that each node to route IP datagrams along the shortest path from itself to the destination specified in the IP datagram.

The size of the link state database and the stability of the network are two important factors that contribute to stable operation of the OSPF protocol. In addition to the number of nodes and links in the network, the size of the link state database is also a function of the number of route prefixes external to the OSPF routing domain whose reachability is shared within the OSPF domain using the OSPF protocol mechanisms. In an unstable network where certain links and/or nodes constantly fail and recover, the operational nodes are forced to constantly exchange information through flooding in order to keep their link state databases synchronized.

OSPF allows for a two level hierarchical network where the logical nodes of this hierarchy are called areas. The root node is called the backbone area. Routing between any two non-backbone areas is always through the backbone area. At the border of any two areas, the topology information of each of the areas is summarized into route prefix reachability information by the border node before flooding this information to the other area. The objective of providing this hierarchical mechanism is twofold. First is to reduce the size of the link state database at each node in the network, and the second is to provide some isolation between stable and unstable portions of the network. The second objective is achieved when instability within a certain area does not affect the summary information for that area.

Recently, there has been an interest in using this link state database to compute explicit paths between edge nodes to support MPLS based traffic engineering. Additional information that needs to become part of the link state database is being defined in extensions to the OSPF protocol. Also, the hierarchy of a network must be chosen carefully so as not to comprose near optimal routing significantly. While TE mechanisms suitable for hierarchical networks are being studied, it is clear that best TE results can be achieved in a single area network.

In addition to the TE issues, area configuration is cumbersome and can be error prone. Inadequate summarization can lead to increased configuration effort without achieving the objectives of splitting into areas. Also, summarization is only applied to the topology information. It is not applied to external prefixes. Therefore, in cases where nodes within an area advertise large number (compared to the size of the area topology) of external prefixes, the size of the link state database may not be significantly reduced.

These issues motivate the objective of this paper which is to describe an OSPF implementation that pushes the limits on the capacity of a node in an OSPF network to be highly scaleable in terms of the size of the network and resilience to instability. Further motivation is provided by recent interest in building extremely high capacity nodes with potentially large number of OSPF speaking interfaces.

The mechanisms described to not change the OSPF protocol itself and rely on the fact that nodes within todays networks are really loosely coupled distributed systems.

II. OSPF Protocol Overview

RFC 2328 is the authauritative source for information on OSPFv2; what follows is a brief overview of the protocol. Details related to the heirarchical aspects of OSPF are omitted in the discussion below. In a hierarchical configuration, some OSPF nodes have an LSDB per area and have the responsibility of limiting flooding of certain LSAs to within the originating area and also have the responsibility of summarizing the information for certain areas.

A. OSPF Interface Types and Speaker Capacity

OSPF considers the following types of interfaces

- point to point interfaces
- broadcast interfaces
- non-broadcast multi-access interfaces
- point to multi-point
- virtual point to point

A packet over SONET (POS) interface on a router that connects to another router within the same OSPF domain is an example of an OSPF point to point interface. An ethernet port over which a router connects to one or more other routers in the same OSPF domain is an example of an OSPF broadcast interface. Non broadcast multi-access (NBMA) interfaces simulate broadcast interface functionality when the underlying physical medium does not support broadcast. OSPF treats broadcast interfaces and NBMA interfaces in very similar terms. In this paper, we do not consider point to multipoint and virtual links. In OSPF, point of multipoint links are treated as being similar to a set of point to point links. Therefore, the proposed mechanism works without any new issues on point to multipoint links. However, its applicability to virtual links needs further study. Figure 1 illustrates the first three types of interfaces.

For broadcast and NBMA interfaces, one of the routers on that network is elected to be a designated router (DR). Only the DR advertises the information about the network while all others just advertise their link to the network. For fault tolerant operation, a backup DR (BDR) is also elected. If a router is neither a DR nor a BDR on an interface, it is expected to participate in the capacity of DROther on that interface.

B. Content at each OSPF Node

Each node in an OSPF network has a link state database (lsdb) which comprises of link state advertisements (lsa). At a given node, these lsas are either self originated, or are obtained from its neighbors using the OSPF protocol. The following types of LSAs are defined. The details of what is contained in each of these LSAs is given in the appendix.

- Router LSA
- Network LSA
- External LSA
- Summary LSA
- ASBR-summary LSA
- NSSA LSA
- Opaque LSA

The router LSAs and the network LSAs together provide the topology of an OSPF area. Each LSA has a standard header that contains advertising router id, LStype, LS id, age, seqnum and a checksum. The LS type, LS id and the

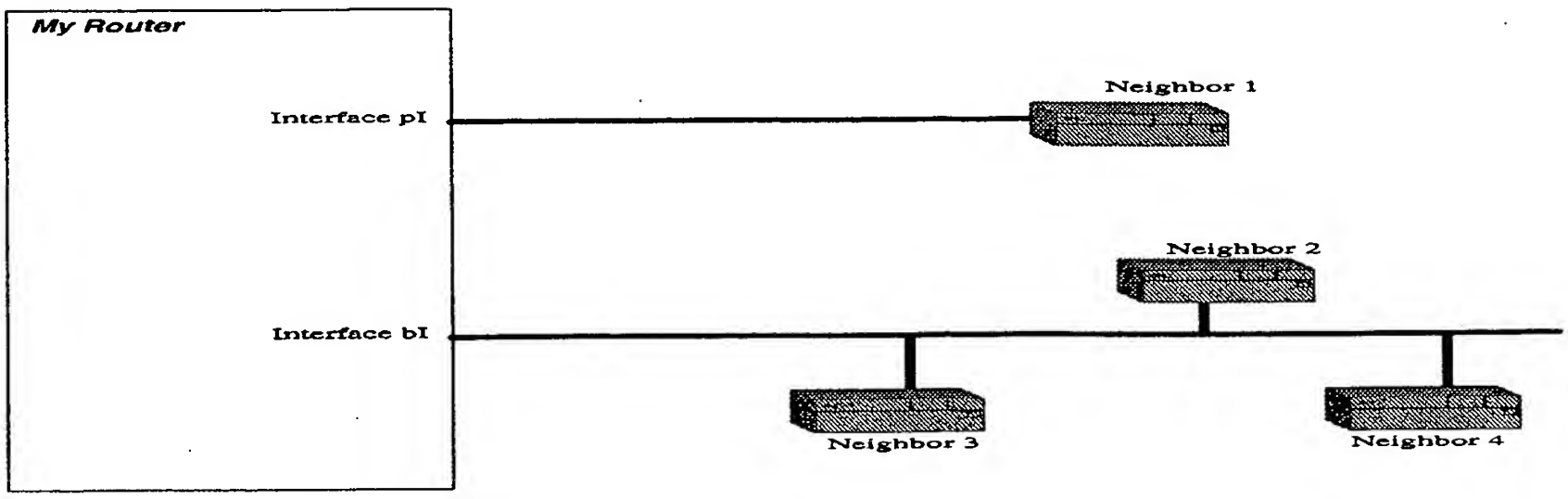


Fig. 1. OSPF Network

Advertising router id together identify an LSA uniquely. For each LSA in the LSDB, the checksum is verified every checkage seconds - if this check fails, something has gone seriously wrong on the node. For two instances of an LSA, the fields age, seqnum and the checksum are used in comparing them.

- The version with the higher sequence number is more recent
- If same sequence number, then the version with the higher checksum is more recent
- If same checksum, then the version with age equals maxage is more recent
- If none of the instances has age equals maxage, if the difference in the age of the two versions is less than maxagediff, the version with the smaller age is more recent
- Otherwise, the two instances are the same

At a given node in an OSPF network, the node keeps the following information

- the link state database (lsdb) - the lsdb comprises of link state advertisements (lsa). LSAs are originated by each node in the OSPF domain and are flooded to every node in the domain. The objective of the OSPF flooding procedure is to keep the lsdbs at all the nodes in the domain synchronized. LSAs are described in detail later in the paper.
- for each interface over which this node is speaking OSPF to one or more neighbors
  - an OSPF interface finite state machine which keeps track of the underlying interface state and the capacity in which OSPF is interacting with its neighbors on this interface - it could be DR, BDR, DROther or P2P. This FSM is shown in figure 2.
  - a neighbor finite state machine for each neighbor that was discovered/configured on this interface - this state machine tracks the state of the communication between this node and the neighbor over this interface. This FSM is shown in figure 3.

Each LSA in the LSDB is aged with time. For self originated LSAs, each LSA is refreshed periodically. When the age of a self originated LSA reaches maxage, the LSA is first flushed out of the OSPF domain by flooding the maxage LSA and then re-originated the LSA with the initial age. For the LSAs originated by other nodes, if the age reaches maxage, they are removed from the LSDB as soon as it is not involved in the process of initial database synchronization with any of its neighbors. If for any reason, a node wants to flush one of its self-originated LSAs from the OSPF domain, it sets its age to maxage and floods it.

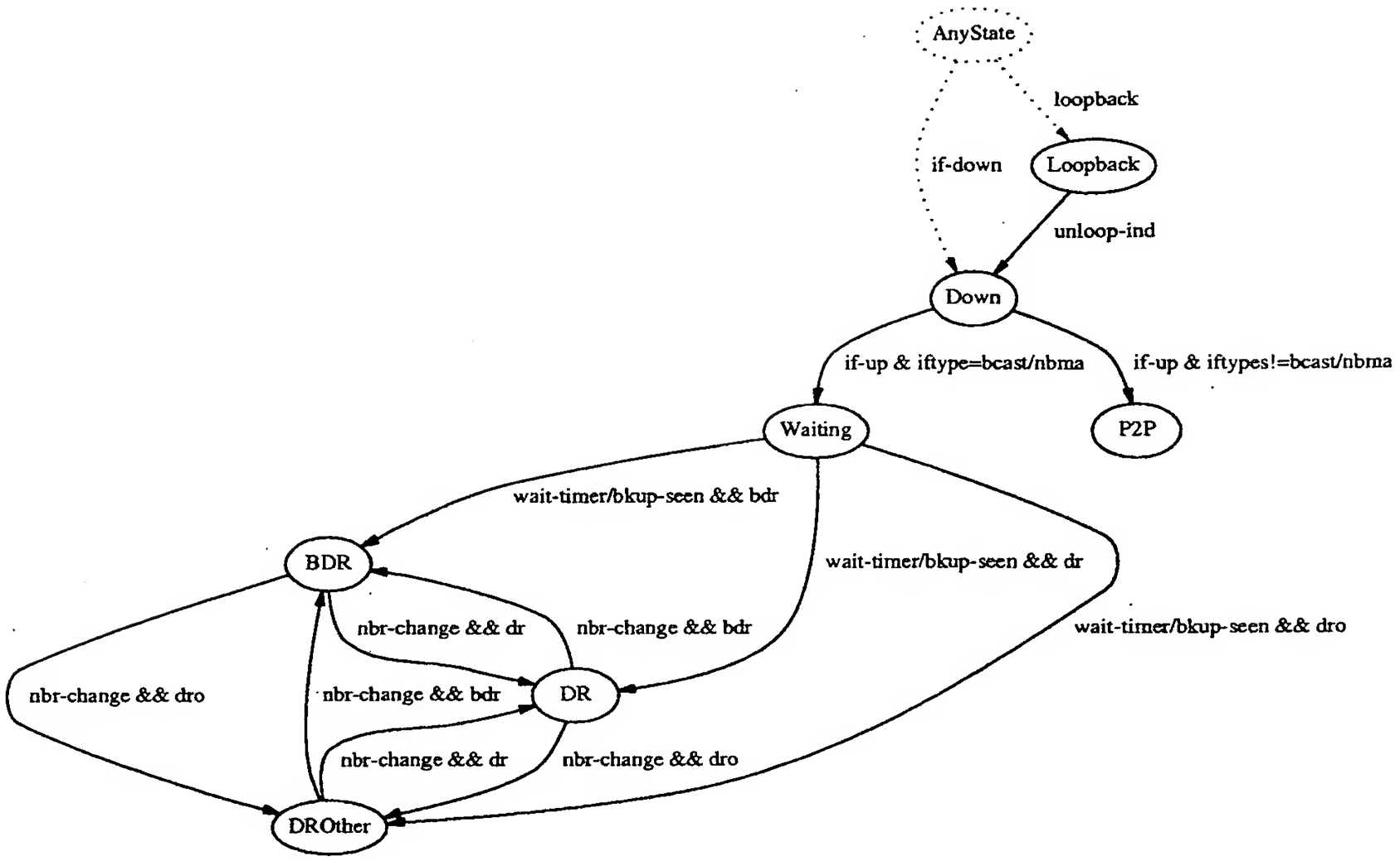


Fig. 2. OSPF Interface FSM

C. OSPF Messages

The following are the types of OSPF messages exchanged between neighbors

- Hello packets
- Database description packets
- Link state request packets
- Link state update packets
- Link state ack packets

The exact use of these messages is included in the following subsections.

D. Establishing and Maintaining Neighbor Relationships

When the OSPF protocol is enabled on an interface, it periodically multicats hello packets on that interface. Hello packets are used to first discover one or more neighbors and where necessary carry all the information to help the DR election process. Among other things, hello packets also carry the identities of all other routers from which the sending node has received hello packets from. When a node receives a hello packet that contains its own identity, the receiving node concludes that bidirectional communication has been established between itself and the sender of the hello packet. When bidirectional connectivity is established, the node decides the capacity in which it must be an OSPF speaker on this interface. At the point when a node must decide whether or not to establish an adjacency with a particular neighbor over one of its interface, the OSPF FSM for that interface would be in one of P2P, DR, BDR or DROther states and the OSPF neighbor FSM for that neighbor would be in state TwoWay. If the decision is not to establish an adjacency, the neighbor FSM stays in state TwoWay. This decision is re-evaluated whenever the OSPF speaker capacity changes.

Once the capcity is established, the two neighboring nodes must decide if they indeed should exchange their lsdbs and keep them in sync. If database exchange needs to be performed, a neighbor relationship (adjacency) is established. For example, if a node is speaking OSPF in the capacity of DROther over an interface, it would decide not to establish an adjacency with another router that is participating as DROther on that interface - DROther speakers only establish adjacencies with DR and BDR speakers.

Once the neighbor relationships (adjacencies) are established and the DR election is done, hello packets are used

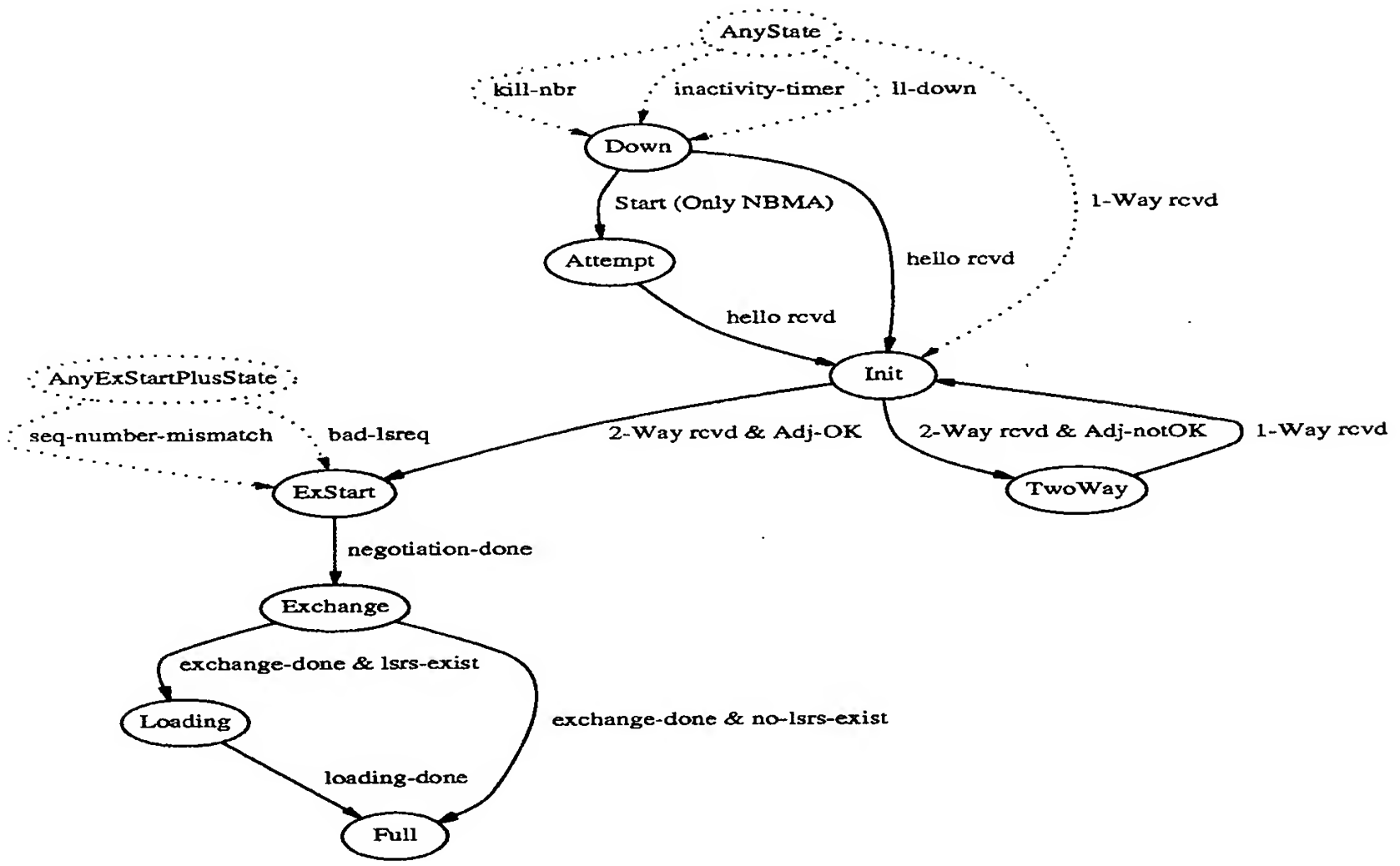


Fig. 3. OSPF Neighbor FSM

as keep-alives for the adjacency and are also used in monitoring any changes that can potentially result in changes in the DR status. Note that a newly identified neighbor can alter the capacity in which the node was speaking on that interface prior to this neighbor being identified. If this is the case, some previously established adjacencies may have to be re-established/terminated.

E. Initial LSDB Synchronization

If the decision is to establish an adjacency, the node is agreeing to keep its lsdb synchronized with its neighbor's lsdb over this interface at all times. At this time the neighbor FSM for this neighbor is in state ExStart. The node enters a master/slave relationship with its neighbor before any data exchange can start. If the neighbor has a higher id, then this node becomes the slave. Otherwise, it becomes the master. When the master/slave relationship is negotiated, the neighbor FSM enters state Exchange.

This lsdb synchornization is achieved in two parts:

- all lsas in the lsdb, except the ones with maxage, at the time of the transition into state Exchange are recorded. This information is summarized and sent to the neighbor in data description packets. When the neighbor receives this summary information, it compares the summary with the contents of its own lsdb and identifies those lsas that are more recent at this node. The neighbor then explicitly requests these more recent lsas by sending link state requests packets to this node. This node then responds to the link state request packets by sending the requested lsas in link state update packets to the neighbor.
- the neighbor is included in the flooding procedure

Note that, in the above, once the summarization of the lsas is done, sending data description packets to the neighbor, responding to link state requests from the neighbor and also including the neighbor in the flooding procedure can all happen concurrently. When this node has sent the whole summary of its lsdb in data description packets to the neighbor and also has received a similar summary from its neighbor, the neighbor FSM transitions into either the Loading of the Full states. It transitions into Loading if it is still expecting responses to the link state request packets it sent to the neighbor. Otherwise, it transitions to state Full. Note that due to the concurrency aspect mentioned earlier, it is possible that all of this nodes link state requests are already responded to even before this node has finished sending all of its

data description packets to its neighbor. In this case, as soon as all the data description packets are sent to the neighbor, the neighbor FSM transitions directly from state Exchange to Full.

When the neighbor FSM transitions to Full, this node includes this interface in the router lsa that it generates.

F. Reliable Flooding Procedure

The flooding procedure is invoked in two scenarios - first is when a node intends to originate/refresh an LSA and second when it receives a new LSA or an update to an existing LSA from its neighbor.

When an updated non self-originated LSA L is received from a neighbor N, one of following scenarios can happen

- No previous instance of L exists in the LSDB; L is a new LSA. If the age of L is maxage and if there are no neighbors in the dbexchange process, then send an ack to N and discard L. Otherwise, timestamp L, ack it and install in the lsdb.
- An older version of L exists in the LSDB. If the older version was received less than minLSarrival time ago, discard L. Otherwise, timestamp L, ack it and install in the lsdb. If there are any neighbors from whom this node is expecting acks for the older version of L, stop expecting such acks.
- A newer version of L exists in the LSDB. Three cases are of interest:
  - If N and this node are still in the dbexchange process, and if N had previously sent a database description packet suggesting that it had a newer version than the one in the LSDB, then this is an error. The db exhange process with N has to start all over again.
  - If the age of the newer version is maxage and its sequence number is maxseqno, then discard L. The intent here is to let the seqno wrap around.
  - If the newer version was received more than minLSinterval time ago, then send the newer version of L to N. Do not expect an ack from N and do not send an ack for L.
- The version in the LSDB is the same as L. In this case, check if this is an implicit ack. If it is an implicit ack, then no need to ack it unless N is the DR. If not treated as an implicit ack, send an ack to N.

If L was installed in the lsdb above, then it needs to be sent to all neighbors except N and other DROther/BDR speakers for which N is the DR (note that if this node was part of more than one area, then the scope of flooding for L would depend on the LSA type of L). In order to ensure reliable delivery of L to its neighbors, L is restransmitted periodically to each neighbor M until an ack is received from M. An ack could be implicit. In the last scenario above, L could be treated as an implicit ack from N if this node was waiting for an ack from N for the version in the LSDB.

When sending L to a neighbor M with which this node is in exchange/loading state, L must be compared with the instance of L that was described in the database description packets sent by M. Two cases are of interest

- L is an older version. In this case, no need to send L to M.
- L is same or more recent. In this case, it is no longer necessary to request the version of L from M. So, stop asking for L when sending link state request packets to M. If M and N are on the same broadcast/NBMA interface and if N is the DR, then it is not necessary to send L to M. In all other cases send L to N.

Note that the above procedure sometimes causes acks to be received even when they are not expected. Such acks are simply discarded.

A maxage LSA L is removed from the LSDB when no acks are expected for L from any neighbor and this node is not in exchange/loading state with any of its neighbors.

In some cases, a node can recieve a self-originated LSA L from one of its neighbors N. If L if more recent than the one in the LSDB (L must have been originated by a previous incarnation of this node), then this node must either flush L by setting its age to maxage and flooding it, or it must originate a newer instance of L with its sequence number being one more than that of L.

G. Use of the LSDB

Whenever the contents of the LSDB change, the routing table is appropriately updated. This may include an SPT computation. In more recently proposed uses of the LSDB, such changes may lead to recomputation of, for example, MPLS explicit paths.

III. A Distributed OSPF Implementation

Our aim is to distribute the OSPF protocol implementation without making any assumptions on 'distributability' of any other routing protocols. For this reason, it is important that the route table computation must be centralized on the main controller card - given that route table computation often requires interaction with information gleaned by other routing protocols and also with provisioned policy information.

Given that the route table computation has to be performed at the controller card, the whole of the LSDB has to be stored on the controller. However, for each LSA, a *delegate port card* is assigned. Therefore, the delegate port card also has a copy of the LSAs that it serves as the delegate for. The delegate is responsible for performing acceptance checks for the LSAs it serves. If an LSA is received by a port card which is not a delegate, that port card just forwards it to a delegate port card if known; otherwise, it sends the LSA to the controller.

Each port card also maintains a copy of the interface FSM and the neighbor FSMs for the interfaces that it owns.

The delegation of LSA processing to port cards is done based on some load balancing heuristic. For example, the total number of LSAs are partitioned equally among all the port cards.

The delegate also performs the checkage and refresh functionality for the LSAs it is handling.

A. *Establishing and Maintaining Neighbor relationships*

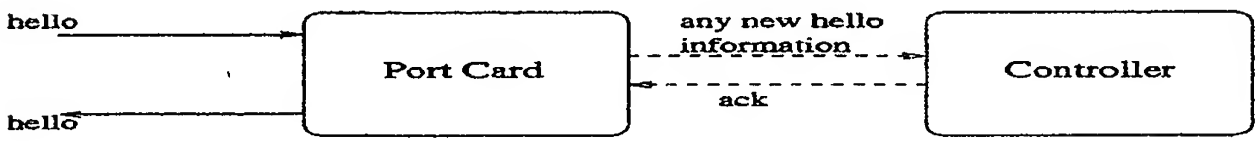


Fig. 4. Distributed Hello Processing

When the OSPF protocol is enabled on an interface, the controller delegates the hello processing and neighbor discovery aspect to the port card that has this interface. Sending and receiving of hello packets is then performed by the port card. Every time a new event for the interface FSM needs to be generated based on incoming hello packet processing, the port card send this event to the controller. This ensures that the port card and the controller have synchronized interface FSMs. The controller, however, does not maintain any timers - it just monitors the functional status of the port

card. All timers are maintained at the port card. Figure 4 illustrates this. Note that the port card expects an ack for the events it sends to the controller.

The port card also maintains a copy of the neighbor FSM for each neighbor discovered through the hello mechanism. The port card is also responsible for executing the DR election procedure. Once the neighbor FSM reaches the state 2 Way, the port card has enough information to decide if it should advance to ExStart. If this is required, the port card sends an event to the controller to initiate the database exchange process. The controller sends an ack for this to the port card when the master/slave negotiation is done.

If the port card is swapped out for any reason during this process, OSPF connectivity on all the interfaces on the port card are considered to be lost and the interface and neighbor FSMs are updates appropriately at the controller. The assumption here is that the controller somehow comes to know about the port card being not available.

B. Initial DB Exchange

Creation of the database description packets requires access to the entire link state database. Therefore, this is done by the controller itself. The controller also sends the link state request packets. However, during the link state request creation process, if it encounters any new LSAs that were not already delegated, it delegates the processing of those to the port cards.

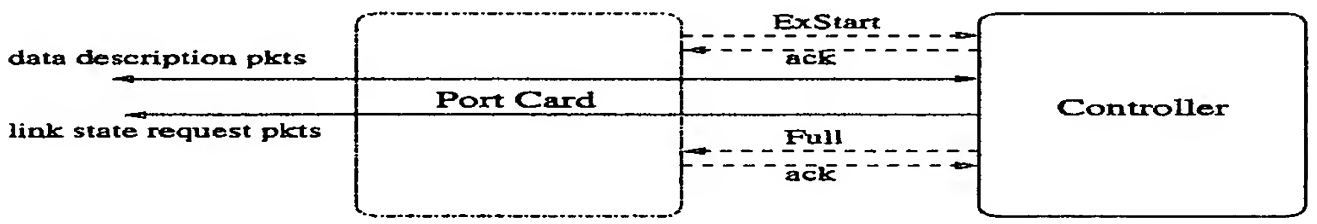


Fig. 5. Initial DB Exchange

When the neighbor FSM at the controller reaches the Full state, the controller sends an event to the port card maintaining a copy of the neighbor FSM to update its state to Full. This is necessary for the port card to make correct flooding scope decisions.

Once again, if the port card is swapped out for any reason during this process, OSPF connectivity on all the interfaces on the port card are considered to be lost and the interface and neighbor FSMs are updated appropriately at the controller.

The initial db exchange process is illustrated in Figure 5.

Note also, that the incoming link state request packets are directly served by the controller. This avoids any age discrepency between the age in the database description packets and the LSA itself.

The load offered on the controller due to processing of data description and link state request packets is not very high because, there can only be one outstanding packet of each of these types per neighbor. Moreover, the number of neighbors simultaneously in the db exchange phase can be limited through configuration.

C. Flooding

When an LSA needs to be sent out from a node to all its neighbors (note that if area design is used, one wouldn't send to all neighbors - scoping the neighbors would be a minor change to the procedure below), the controller initiates this by broadcasting the LSA to all the port cards. Each port card then sends the LSA to the appropriate neighbors connected on through the interfaces on that port card as part of the link state update packets.

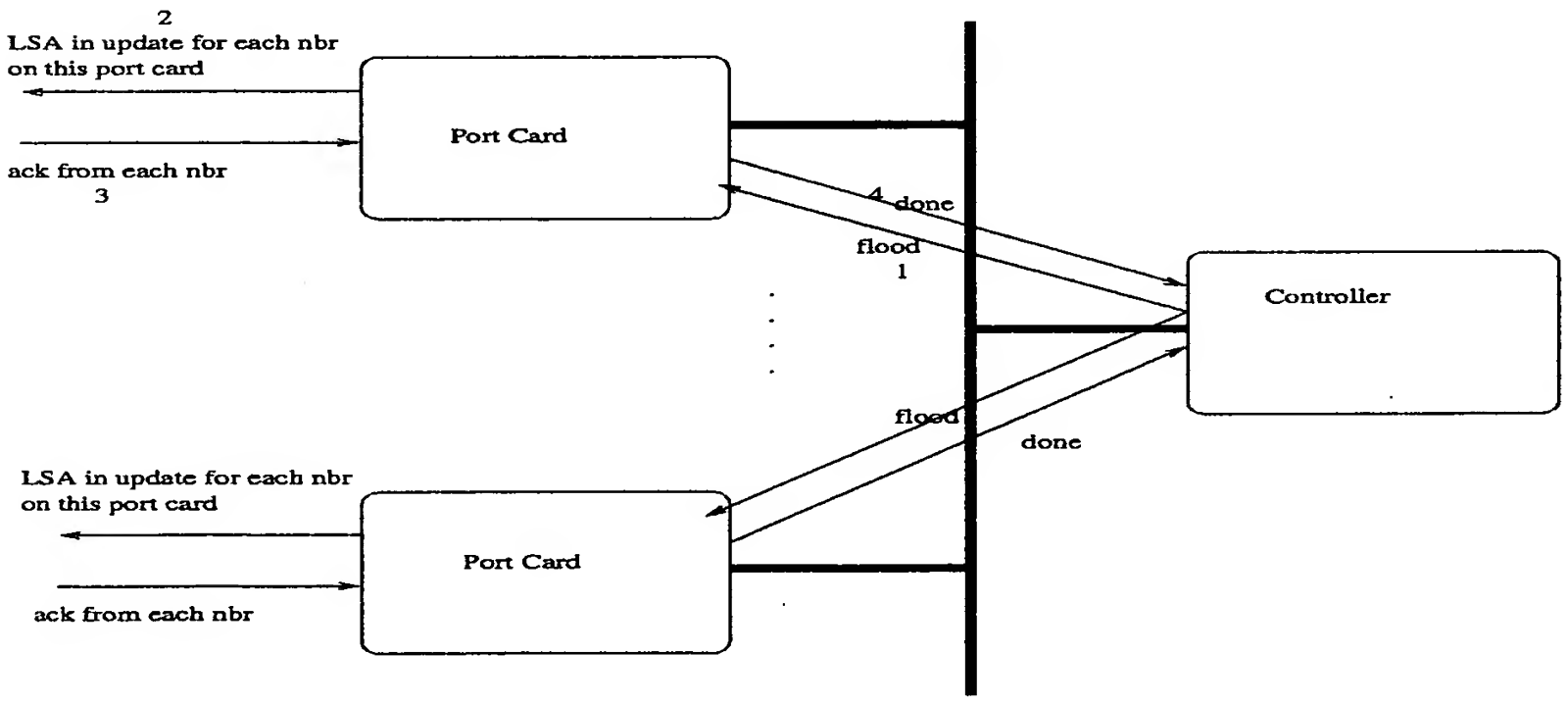


Fig. 6. Distributed Flooding Functionality

The port cards handle all the issues of retransmission and acknowledgement related logic including implicit acknowledgements. When the port card receives acks from all the neighbors, it sends a done event to the controller. This procedure is illustrated in Figure 6.

Note that it is assumed that the flood and done events are exchanged between the port cards and the controller in a reliable manner. The flooding of an LSA is considered complete when all port cards respond with a done event.

LSAs coming in to a node are received in link state update packets. The receiving port card checks the LSU validity and then extracts each LSA from it. For each LSA extracted, the port card checks if it is the delegate for it. If not, it forwards the LSA to the delegate port card - if a delegate is not known the LSA is forwarded to the controller. The port card does not maintain any state for LSAs that it is not the delegate for.

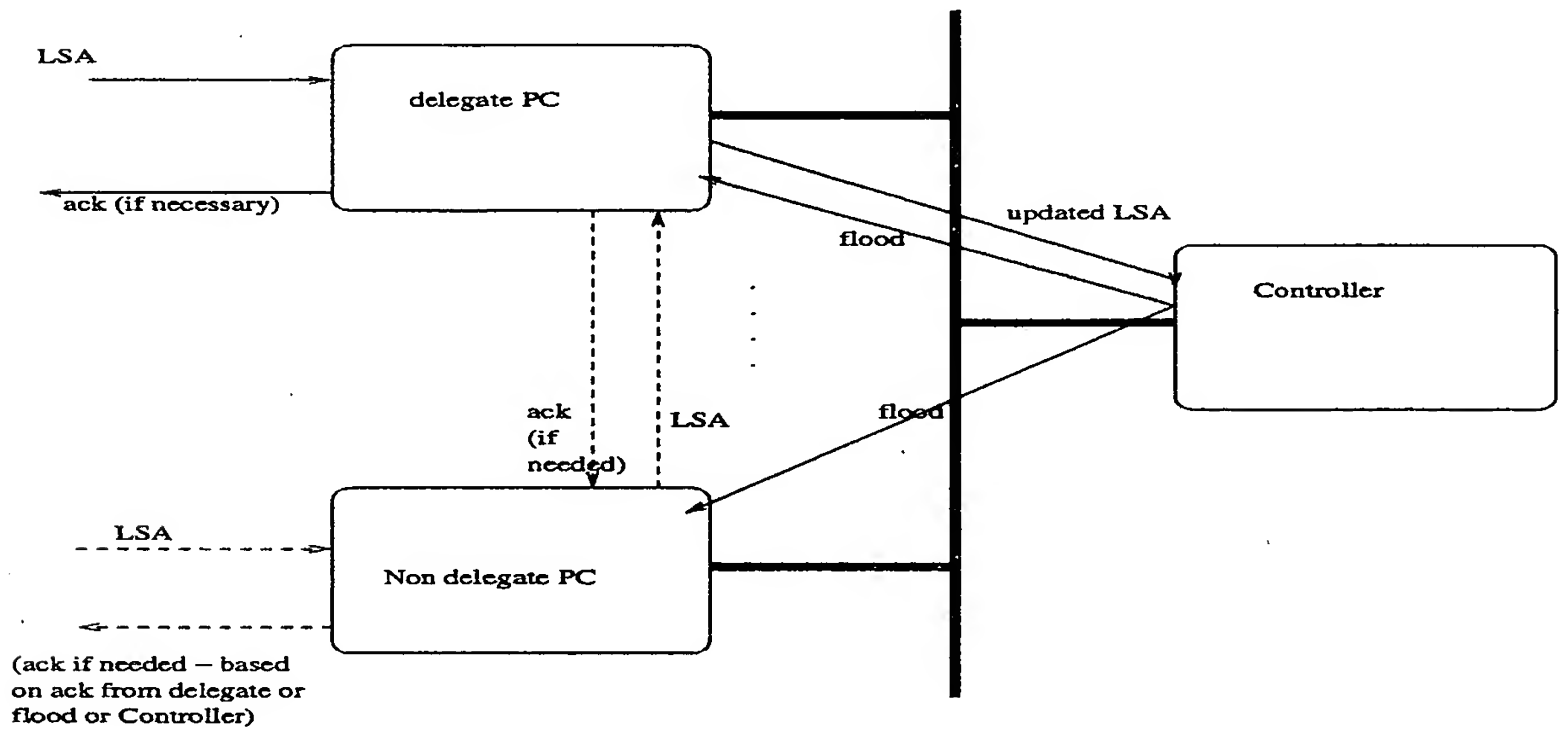


Fig. 7. Distributed Processing of Incoming LSA Updates

If it is the delegate, it checks if the age for the LSA is maxage. If so, it ceases to be the delegate and forwards the LSA to the controller. Otherwise, it checks if this LSA should be accepted. If not, it just sends an ack to the sending neighbor if necessary. Otherwise, this LSA needs to be accepted and is forwarded to the controller. The delegate port card does not send an ack to the neighbor until the controller decides to flood that LSA. The flood event acts as an implicit acknowledgment that the controller has received at least the required version of the LSA. When the flood event is received, the port card decides if an explicit ack needs to be sent to the neighbor. If so required, it sends the ack. Note that sending of the LSA to the controller does not require reliable communication. If it doesn't get through, the neighbor will time out and resend anyway.

A delegate can get an LSA from another port card. In this case, the processing is same as above, except that the ack is sent through the port card that originally received the LSA. If the LSA is accepted by the delegate and passed on to the

controller, a minor optimization to the above is possible. The port card that originally received the LSA can send back an ack to the sending neighbor based on the flood from the controller. Once again, note that no reliable communicated is needed in this scenario.

Procedures for handling incoming LSAs is illustrated in Fig 7.

Also, when self-originated LSAs need to be refreshed, the delegate port card informs the controller that the LSA needs to be re-originated. The controller then floods a new instance of the requested LSA and the delegate updates its copy with the new one. The indication from the delegate port card to the controller has to be reliable. If the number of self originated LSAs is small then the controller itself may take responsibility of keeping track of the last refresh time. If this responsibility is indeed delegated to a port card and the port card dies, then the controller must compare the time of death of the port card with the timestamp of self originated LSAs for which the dead port card was a delegate for. The controller may either delegate to another port card indicating the remaining time for the refresh, or postpone this delegation until the next refresh and keep track of the refresh timer itself. In addition to taking over the refresh functionality, the controller must also handle the checkage functionality.

D. The Issue of LS Age (have a more optimal solution).

For the correctness of the procedures described above, it is essential that the ageing of LSAs is done consistently among the controller and the delegate port cards. This is because the LSA acceptance procedure for incoming LSAs depends on the comparison of the age of the incoming LSA and that of the LSA existing in the LSDB.

To achieve this consistency, the controller floods a timer tic to all the port cards. The port cards use this tic in updating the age of their copies of the LSAs. This ensures that the age of the LSA on the port card is always less than or equal to that on the controller. Because of this, if a delegate port card dies and the controller has to take over the responsibility of an LSA, the situation would be no worse than a temporary clock speedup at an OSPF node and the comparisons for any subsequent retransmissions from a neighbor would be consistant with the previous comparisons performed by the dead delegate.

Note that the above procedure can be made more robust my requiring each port card to ack after every x number of ticks for some allowable drift of x seconds. The OSPF protocol itself is robust up to drift of 15 minutes and does not require participating nodes to keep their clocks synchronized.

E. Further Optimizations

Before an LSA update is sent from a delegate port card to the controller, the LSA can be preprocessed and presented in a form where the controller spends much less cpu time in processing it. Also, LSA updates can be sent in batches to reduce the number of messages.

One example of preprocessing is the following. Given the way router LSAs are structured, for each link described in it, the node reachable using that link has to be searched from the set of all the router LSAs using the router id. As the number of nodes and links increases, an SPT computation may require m searches, one for each link, on a set of n nodes. In the best case, each of these m searches is O(log n). Preprocessing can be performed to make this O(log n) operation into O(1). The search overhead is now on the delegate port card and is distributed among a number of port cards. Note that while this overhead is not significant for infrequent SPT computation, it could become significant in an unstable network.

F. Claims

Claim 1: A database description packet from a neighbor is never lost as long as the port card with the physical interface connected to that neighbor is alive.

Claim 2: The interface and neighbor FSMs on the port card and the controller are always synchronized.

Claim 3: The procedure is robust in the presense of hot swappable port cards.

Claim 4: The total amount of memory used (sum of memory used by all port cards and the controller) for the LSDB is at most twice the amount used in the controller itself.

Claim 5: The OSPF traffic traversing the internal bus connecting the port cards and the controller is never more than twice the centralized version and could be much less on the average. (needs further work to get some reasonable bounds)

IV. Dealing with very large areas

If the whole AS of an ISP is treated as a single area then it is conceivable that although each node can handle the flooding load using our distributed implementation, link oscillations on some corner of the network may lead to updates of routing tables across all the nodes.

This can be achieved by making the SPT computation algorithm smarter. For example, the topology could be partitioned into connected components ...

V. Conclusion

We have described a distributed implementation of the OSPF protocol in a network node with a loosely coupled distributed hardware architecture. The following functionality is distributed to the port cards

- hello processing and DR election
- dealing with incoming flood of LSA updates
- preprocessing of LSAs
- ~~checkage~~ and refresh functionality
- all retransmission timers and ack processing for LSA updates

This procedure significantly lowers the burden on the controller card and is robust in the presense of failing/hot swappable port cards.

References

[1] J. Moy, "OSPF version 2," Request for Comments 2328, Internet Engineering Task Force, Apr. 1998.

Appendix

```
LSA Types:
----------

RTR LSA: (LS type = 1)
  identfied by: RTR id
  contains: links and their ip addresses / ifindex
  contents:
     LS ID = rtr id
     Adv RTR = rtr id
     indication of whether ABR and/or ASBR (bits B and E, respectively)
     num of links
     foreach link
        link type + link-id + link-data
        ( 1 ptpt + nei rtr id + IP addr of interface/ifindex,
          2 link2transnet + IP addr of DR interface + IP addr of interface,
          3 link2stubnet + IP addr of interface + IP mask,
          4 vlink + nei rtr id + IP addr of interface)
        link cost (non-zero except for stub links)

NW LSA: (LS type = 2)
  identified by: IP addr of DR interface connected to this network
  contains: rtr ids of all rtrs connected to this network including the
            DR itself (only those that are fully adjacent)
  contents:
     LS ID = IP addr of DR interface
     Adv RTR = DRs rtr id
     Network Mask
```

```
        attached rtr id 1
        attached rtr id 2
        attached rtr id 3
        ...

Summary LSA: (LS type = 3)
  identified by: RTR id of ABR
  contents:
    LS ID = IP prefix of network being summarized
    Adv RTR = rtr id of ABR
    metric/cost
    network mask

ASBR Summary LSA: (LS type = 4)
  identified by RTR id of ASBR
  contents:
    LS ID = rtr id of ASBR
    Adv RTR = rtr id of ABR
    metric/cost

External LSA: (LS type = 5)
  (higher OSPF rtr id of adv rtr used as tie breaker if prefix, cost and
   forwarding address are all the same)
  (type 1: internal cost + external cost)
  (type 2: external cost only)
  identified by RTR id of ASBR
  contents:
    LS ID = IP prefix of external network
    Adv RTR = rtr id of ASBR
    metric type (E bit == 1 ===> type 2 metric)
    metric/cost
    Forwarding address (typically same as rtr id of ASBR)

NSSA LSA:
   (* TBD *)

Identifying an LSA:
  unique key is (LS type, LS id, Adv rtrid)

LSA accept decision:
Comparing two instances of an LSA:
  seq number, age and checksum are used to determine which is more recent
  -- higher seq number, if same,
  -- larger checksum, if same,
  -- one with maxage, if none
  -- if diff in age is less than maxagediff, pick smaller age
  -- else, they are same, so do not accept updated lsa
```

A Distributed OSPF Implementation

Pramod Koppol

Bell Labs Research
Lucent Technologies
pramod@lucent.com

8/21/01 — work began sometime March/April

Presentation Outline

- Background and Motivation
- What to distribute and why
- The method
- Summary

Background

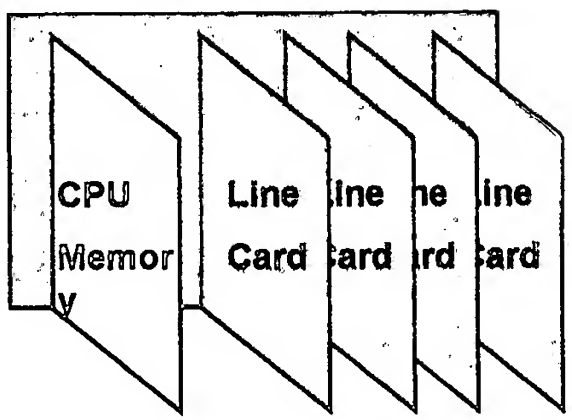
Bell Laboratories

- **OSPF is a widely deployed intra-AS link state IP routing protocol**
  - → Uses reliable flooding mechanism to disseminate advertisements
  - → Three main functions are: Flooding, SPT computation and routing table update, and neighbor maintenance
  - → Recent work on doing SPT computation "cleverly"
  - → Supports a 2-level hierarchy to localize flooding and faults
- **There is a lot of interest in intra-AS Traffic Engineering**
  - → Extensions being made to OSPF to support TE
  - → Increase in the information exchanged by OSPF
  - → Possibly more frequent SPT computation
  - → Works best when configured as a single OSPF area
- **Evolution of router architectures**
  - → Potentially support thousands of interfaces
  - → Have a lot of processing power

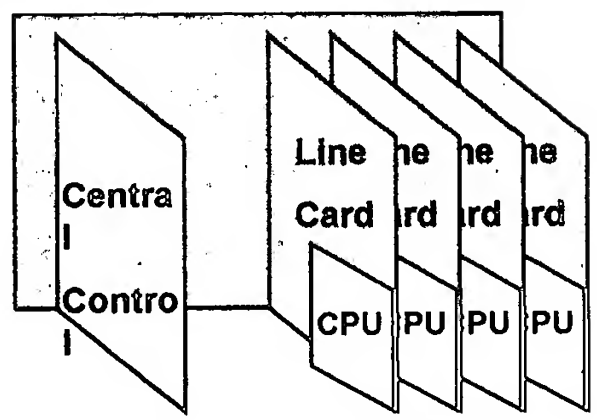
Evolution of Packet Switch Architectures

Bell Laboratories

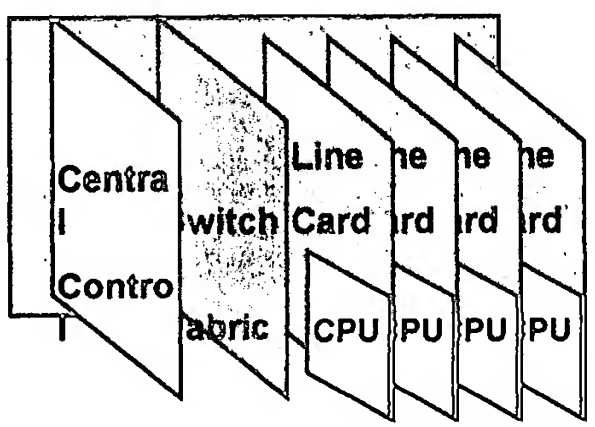
First Generation
Single CPU – multiple line cards
Single electrical backplane



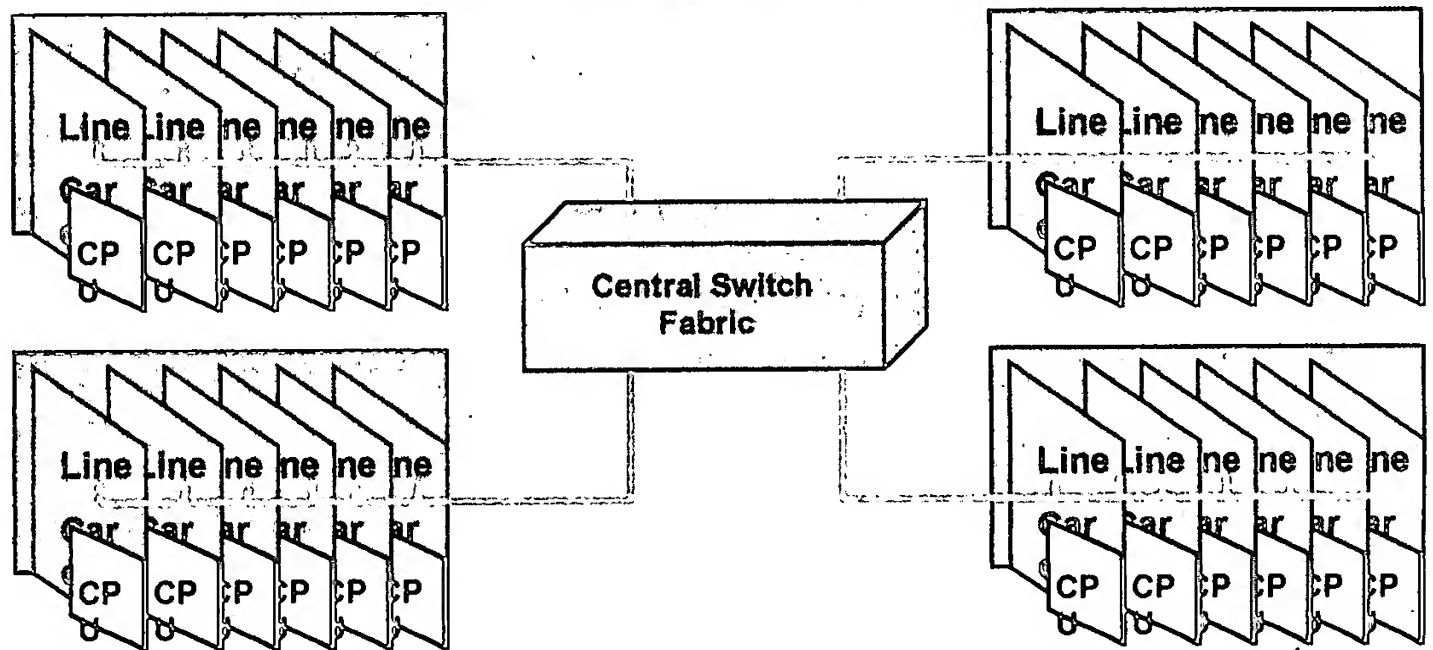
Second Generation
One CPU per Line Card
Central Controller for Routing Protocols



Third Generation
One CPU per Line Card
Central Controller for Routing Protocols
Switch Fabric for inter-connection



Fourth Generation
Multiple shelves of Line Cards
Centralized Switch Fabric
Optical links interconnecting Line Cards and Fabric



Motivating aspects

- Routers with large number of interfaces and high processing power
- Increasing interest in TE
- Recent interest in millisecond convergence – possible with sub-second hellos and frequent SPT computation
- Configuring a large number of areas can increase the potential for human error

- What can be done?
  - ➔ Current router architectures scale the forwarding capacity by distributing the forwarding functionality to the line cards
  - ➔ Line cards usually have a reasonably powerful CPU that is most likely under-utilized
  - ➔ All control software runs on a single controller card

- ***Why not distribute some software functionality to the line cards also?***

Major Functions of OSPF

- Flooding
  - → **Send info over every interface belonging to the same area**
  - → **Duplicate info can be received over every interface belonging to an area**
  - → **A number of timers may need to be maintained to achieve reliable flooding**

- Maintenance functions
  - → **Sending and receiving hello messages to/from neighboring nodes**
  - → **Performing leader election for broadcast interfaces**
  - → **Periodic checksum verification**

- SPT Computation & Routing table update
  - → **Routing table update needs to be centralized unless other protocols are also appropriately distributed**
  - → **SPT computation by itself may not be a very big issue; but frequent computations can add up in cost**
  - → **Incremental SPT computation techniques have been shown to significantly lower CPU load**

What functionality can be distributed to the line cards ?

- **Receiving of LSAs**
- **Reliable flooding**
- **Hello processing & leader election**

- **Requirements**
  - ➔ Should work with hot swappable line cards
  - ➔ Should not make any changes to the protocol itself
  - ➔ Should not require a complete rewrite of existing software

- **The method**
  - ➔ See attached paper/draft

Can be applicable to PNNI & IS-IS & VNN

EXHIBIT B

Lucent Technologies
Bell Labs Innovations

subject: **Patent Submission IDS 124380**
"Scaling OSPF To Very Large Areas"

date: **August 21, 2001**

from: **Matthew J. Hodulik**
Intellectual Property-Law
HO 3K-223 (732) 949-9742

P. Koppol:

Patent submission #124380 was formally docketed to consider the patentability of the above-identified subject matter. You are identified as the originator.

Should you have any questions regarding the subject matter, please feel free to contact me.

Matthew J. Hodulik
Corporate Counsel

Copy to:
T. V. Lakshman

SUBMISSION NO. : 124380
ATTORNEY : Hodulik, Matthew J.
TITLE :

Scaling OSPF To Very Large Areas

----------MAIN INFORMATION----------

ITEM STATUS	: Open	**LUCENT RATING**	: II
STATUS DATE	: 8/21/01	**GOVT. CONTRACT**	: N
OPEN DATE	: 8/21/01	**TYPE**	: Patentability
CLOSE DATE	:	**DEADLINE DATE**	:
CLASS CODE	: II	**TECHNOLOGY**	: ATM/Data Networking
BU CODES(S)	: DNET	**OWNER**	: Lucent

FOREIGN FILING RECOMMENDATION:

----------EURO SUBMISSION REVIEW INFORMATION----------

CURRENT STATUS	:	**ACTION TAKEN BY**	:
PATENT LIAISON	:	**PRIORITY CODE**	: 0

----------SUBMITTER INFORMATION----------

SUBMITTER NAME	:	Koppol, Pramod V.N.
COMPANY	:	LUCENT
LOCATION	:	HO4C638x3824
EXTENSION	:	HO4C638x3824
DEPARTMENT	:	JK125A000
DIRECTOR	:	S. Sataluri

Brief Description: